Formal Design Analysis Framework: an Aspect-Oriented Architectural Framework

Dissertation Defense
Lirong Dai

Ph.D. Supervisory Committee: Dr. Kendra Cooper (Supervisor)
Dr. Lawrence Chung
Dr. Gopal Gupta
Dr. I-Ling Yen

The University of Texas at Dallas, USA
October 19, 2005
Outline

- Motivation
- Research Problem
- Related Work
- Solution: Formal Design Analysis Framework (FDAF)
- FDAF Approach Illustrations
- Summary
- Contributions to Knowledge
- Conclusions and Future Work
Motivation

- Software development goal:
  - Deliver complete, consistent, and correct products with low time and cost

- Challenges presented:
  - Systems continue to increase in size and complexity
  - Realizing non-functional requirements
    - Functional requirements: functions and tasks a system must support
    - Non-functional requirements: constraints on these functions and tasks, important factor to decide a system's success (e.g., performance, security, reliability, etc.)

- Investigation on software development methodologies
  - Software development life cycle

Non-functional properties design and analysis at software architecture level
Motivation

- **Software architecture**
  - Structure of a system, including: *software elements, elements’ external visible properties, relationships among elements* [Bass L.]
  - Represents the earliest design decisions, hardest to change later, critical to be right
  - Strongly impacts final product’s quality, “The right architecture paves the way for system success”, [Shaw & Garlan]

- **Architectural design of non-functional properties**
  - Realizes a system’s non-functional requirements

- **Architectural analysis of non-functional properties**
  - Enforces an assessment step at an early stage to discover and fix defects
  - Software design flaws cost the economy an estimated $59.5 billion annually [NIST]

![Cost of Fixing Errors](Figure adapted from: Scott W. Ambler, “Examining the Cost of Change”, Agile Modeling Essay)
Motivation

- An approach supports the effective design and analysis of non-functional properties for software architecture is needed:
  - Improve the readability and understandability of the design
    - Visual modeling is helpful
      - ✓ Unified Modeling Language (UML) is becoming a de-facto standard
      - ✗ Semi-formal, no automated analysis supported
  - Provide automated analysis to improve the quality of the design
    - Accurate, reliable analysis results, reduce the effort/human error/cost
      - ✓ Formal methods support this, e.g., Architectural Description Languages (ADLs), Promela
      - ✗ Difficult to read, write, understand, and modify designs
  - Support the maintainability and evolvability of the design
    - Many sources of change, maintenance accounts for 50% of total development cost
      - ✓ Aspect-oriented paradigm supports this
        - Adaptation of aspect-oriented programming principles to the design phase
        - Tangling concerns encapsulated in aspects
        - Aspect added one at a time, each aspect model can be developed independently, reduce maintenance time and cost when change happens
        - Weaving process to generate a complete design if necessary
Research Problem

- Represent non-functional properties in a software architecture
- Model crosscutting non-functional properties in UML
- Automatically analyze a UML based architecture design using existing formal methods
- Support the maintainability and evolvability of an architecture design with non-functional properties
Related Work

- Non-functional property design and analysis work in the literature
  - Focuses on performance and security
  - Approaches classified as: Semi-formal, Formal, Integrated semi-formal and formal, Aspect-oriented

- Comparisons of semi-formal approaches with the FDAF

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NFP Architectural Representation</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Modeling Notations</td>
<td>UML</td>
<td>UML</td>
<td>UML</td>
<td>UML</td>
<td>UML</td>
<td>UML, formal languages</td>
</tr>
<tr>
<td>Analysis</td>
<td>Not supported</td>
<td>Manual analysis</td>
<td>Automated simulation</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Automated analysis</td>
</tr>
<tr>
<td>Maintainability and evolvability of Design</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Supported using aspect-oriented paradigm</td>
</tr>
<tr>
<td>Activity Level</td>
<td>Architecture</td>
<td>Architecture</td>
<td>Detailed design</td>
<td>Architecture</td>
<td>Detailed design</td>
<td>Architecture</td>
</tr>
</tbody>
</table>
## Related Work

- Comparisons of formal approaches with the FDAF

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NFP Architectural Representation</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Modeling Notations</td>
<td>Petri Nets</td>
<td>Colored Stochastic Petri Nets</td>
<td>Petri Nets, Temporal Logic</td>
<td>Alloy</td>
<td>Discrete Time Labeled Transition System</td>
<td>Z</td>
<td>UML, formal languages</td>
</tr>
<tr>
<td>Analysis</td>
<td>Automated simulation</td>
<td>Automated analysis</td>
<td>Not supported</td>
<td>Automated analysis</td>
<td>Automated analysis</td>
<td>Automated analysis</td>
<td>Automated analysis</td>
</tr>
<tr>
<td>Maintainability and Envolvability of Design</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Supported using aspect-oriented paradigm</td>
</tr>
<tr>
<td>Activity Level</td>
<td>Detailed design</td>
<td>Detailed design</td>
<td>Architecture</td>
<td>Architecture</td>
<td>Detailed design</td>
<td>Architecture</td>
<td>Architecture</td>
</tr>
</tbody>
</table>
## Related Work

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Criteria</th>
<th>UML/Æmilia ADL Approach</th>
<th>UML/Petri Nets Approach</th>
<th>UML/Theorem Prover Approach</th>
<th>UML/Promela Approach</th>
<th>FDAF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NFR</strong></td>
<td>Performance</td>
<td>Performance</td>
<td>Security</td>
<td>Security</td>
<td></td>
<td>Performance</td>
</tr>
<tr>
<td><strong>NFP</strong></td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td><strong>Architectural Representation</strong></td>
<td>Not supported</td>
<td>UML, Petri Nets</td>
<td>UML, First-Order Logic</td>
<td>UML, Linear Time Temporal Logic</td>
<td>UML, formal languages including Rapide, Armani, Æmilia Promela, Alloy</td>
<td></td>
</tr>
<tr>
<td><strong>Modeling Notations</strong></td>
<td>UML, Æmilia</td>
<td>UML, Petri Nets</td>
<td>UML, First-Order Logic</td>
<td>UML, Linear Time Temporal Logic</td>
<td></td>
<td>UML, formal languages including Rapide, Armani, Æmilia Promela, Alloy</td>
</tr>
<tr>
<td><strong>Analysis</strong></td>
<td>Automated analysis</td>
<td>Automated analysis</td>
<td>Automated analysis</td>
<td>Automated analysis</td>
<td>Automated analysis</td>
<td>Automated analysis</td>
</tr>
<tr>
<td><strong>Automated Translation from UML to Formal</strong></td>
<td>Not supported</td>
<td>Not supported</td>
<td>Supported</td>
<td>Supported</td>
<td>Supported</td>
<td></td>
</tr>
<tr>
<td><strong>Maintainability and Evolvability of Design</strong></td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Supported using aspect-oriented paradigm</td>
<td></td>
</tr>
<tr>
<td><strong>Activity Level</strong></td>
<td>Architecture</td>
<td>Detailed Design</td>
<td>Detailed Design</td>
<td>Requirement</td>
<td>Architecture</td>
<td></td>
</tr>
</tbody>
</table>
### Related Work

- Comparisons of aspect-oriented approaches with the FDAF

<table>
<thead>
<tr>
<th>Approaches</th>
<th>COMQUAD Component Model</th>
<th>Re-QoS Management Method</th>
<th>Aspect-Oriented Secure Application</th>
<th>FDAF</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NFP</strong></td>
<td>No specific NFP selected, general approach</td>
<td>No specific NFP selected, general approach</td>
<td>Security</td>
<td>Performance Security</td>
</tr>
<tr>
<td><strong>NFP Architectural Representation</strong></td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Supported</td>
</tr>
<tr>
<td><strong>Modeling Notations</strong></td>
<td>UML</td>
<td>Not supported</td>
<td>UML</td>
<td>UML, formal languages</td>
</tr>
<tr>
<td><strong>Analysis</strong></td>
<td>Not supported</td>
<td>Not supported</td>
<td>Not supported</td>
<td>Automated analysis</td>
</tr>
<tr>
<td><strong>Maintainability and Evolvability of Design</strong></td>
<td>Aspect-oriented approach used, but did not address how to use the approach to support this</td>
<td>Aspect-oriented approach used, but did not address how to use the approach to support this</td>
<td>Aspect-oriented approach used, but did not address how to use the approach to support this</td>
<td>Supported using aspect-oriented paradigm</td>
</tr>
<tr>
<td><strong>Activity Level</strong></td>
<td>Detailed design</td>
<td>Implementation</td>
<td>Implementation</td>
<td>Architecture</td>
</tr>
</tbody>
</table>
Solution:

**Formal Design Analysis Framework**

- FDAAF solutions to the research problem:
  - Represent non-functional properties in software architecture
    - Aspects, definitions provided for aspects with guidance on using aspects
  - Model non-functional properties in UML
    - A UML extension is proposed, defined with syntax and semantics
  - Automatically analyze a UML based architecture design using existing formal analysis tools
    - Automated formalizing (part of) UML into a set of formal languages
  - Support maintainability and evolvability of an architecture design with non-functional properties
    - Aspect-oriented paradigm
Formal Design Analysis Framework

- Overview
- FDAF Aspects
- UML extension for aspect modeling
- Automated analysis of UML aspect architecture design
- Validation
- Case study: Domain Name System (DNS)
FDAF Overview

**Inputs:**
- Semi-formal UML Design Model
- Functional and Non-functional Requirements

**Aspect Repository**
- Performance Aspect: Response Time, Resource Utilization, Throughput...
- Security Aspects: Data Origin Authentication, Role-Based Access Control, Log for Audit...

**Aspect-Oriented Formal Models**
- Response Time
- Resource Utilization
- Throughput
- Data Origin Authentication
- Role-Based Access Control
- Log for Audit

**Existing Formal Analysis Tools**
- Rapide
- Armani
- Æmilia
- Promela
- Alloy
- Promela

**Outputs:**
- Baselined Semi-formal, Formal Design Models
- Analysis Results Generated by Formal Analysis Tools

**Stakeholders:**
- Architects
- Designers
- Requirement Engineers

**Analysis Results**
FDAF Aspects

- **Property aspects**
  - Describe properties of systems using certain format of data (e.g., numerical values)
  - Do not trace to specific code modules
  - Defined using UML stereotypes with tags
  - E.g., performance aspects, including response time, resource utilization, throughput, etc.

- **Substantive aspects**
  - Provide additional capabilities
  - Constructed in code
  - Defined using UML class diagram (static view) and sequence diagram (dynamic view)
  - E.g., security aspects, including data origin authentication, role-based access control, log for audit, etc.
FDAF UML Extension for Aspect Modeling

- Introduce a new graphical icon, a parallelogram notation, to present aspects in a UML design
- Two modeling mechanism for property aspects and substantive aspects
  - Property aspect modeling mechanism
    - Annotates a property aspect to a UML model element
    - Weaving (concrete syntax):
  - Substantive aspect modeling mechanism
    - Generates an aspect-oriented design that can be mapped into aspect-oriented implementation, e.g., AspectJ
    - Abstracts aspect concepts join point, advice etc., up to design level with syntax and semantics (UML action semantics and OCL)
    - Weaving (concrete syntax):
High-Level Package View of the UML Extension for Aspect-Oriented Modeling
(Substantive Aspects)
FDAF Automated Analysis of UML Aspect Architecture Design

- Architecture design defined in UML class diagram and UML swim lane activity diagram, then extended with aspects
- Formalizing this part of UML into formal languages using translational semantic approach [Meyer]
  - Algorithm defined and verified with correctness
  - Algorithm implemented for the automated translation
  - Using existing formal analysis tools
  - Supported formalization include:
    - Rapide architecture description language
    - Armani architecture description language
    - Æmilia architecture description language
    - Promela
    - Alloy
FADF Validation

- All translation algorithms have been proven with correctness

- FDAF tool support
  - Extended on open source ArgoUML
    - Implementation language Java
    - Extension: around 110 Java classes, 9250 line source code
  - Extension include:
    - Aspect repository with a set of predefined aspects
    - Weaving an aspect into a design
    - Automatically generate Rapide, Armani, Æmilia, Promela, Alloy for an extended UML architecture design
    - Interfaces with formal analysis tools
  - Source code available upon request
FDAF Case Study: Domain Name System

DNS Functionalities:
1. Convert host name \texttt{www.cnn.com} to IP address “207.25.71.28”
2. Convert IP address “207.25.71.28” to host name \texttt{www.cnn.com}
FDAF Approach Illustration Overview

**Inputs:**
- Semi-formal UML Design Model
- Functional and Non-functional Requirements

**Aspect Repository**
- Performance Aspect: Response Time, Resource Utilization, Throughput...
- Security Aspects: Data Origin Authentication, Role-Based Access Control, Log for Audit...

**Extended UML Design Model**
- Performance Module
- Security Module
- FDAF Translation Tool

**Analysis Results**

**Outputs:**
- Baseline Formal Design Models
- Analysis Results Generated by Formal Analysis Tool

**Stakeholders:**
- Architects, Designers, Requirement Engineers

**Aspect-Oriented Formal Models**
- Response Time
- Resource Utilization
- Throughput
- Data Origin Authentication
- Role-Based Access Control
- Log for Audit

**Existing Formal Analysis Tools**
- Rapide
- Armani
- Æmilia
- Promela
- Alloy
- Promela
FDAF Approach Illustrations
(Definition, design and analysis of supported aspects)

1. Extend the design with an aspect
2. Automatically generate formal specification
3. Analyze formal specification using the corresponding formal analysis tool
FDAF Approach Illustration: Resource Utilization Performance Aspect

- Property aspect
- Aspect definition
  - Resource utilization aspect for systems
    - <<sArrivalRate>>, arrival rate of jobs
    - <<sNetworkPop>>, mean numbers of jobs
    - <<sNetworkResponse>>, mean response time
  - Resource utilization aspect for components
    - <<sServiceTime>>, service time
    - <<sVisits>>, percentage of jobs visit
    - <<sReplication>>, number of component copies
    - <<sUtilization>>, busy percentage time;
    - <<sLength>>, queue length
    - <<sResponseTime>>, response time
  - Resource utilization aspect for connectors
    - <<sDelayTime>>, transmission delay time
    - <<sVisits>>, percentage of jobs visit
FDAF Step One: Extend DNS architecture design with resource utilization aspect
FDAF Step One: Extend DNS architecture design with resource utilization aspect (continued)
FIDA F Step Two: Automatically generate Armani formal specification

- Armani
  - Monroe, Carnegie Mellon University
  - First-order predicate logic-based language
  - Underlying performance analysis theory: queueing network theory

- Mapping from extended UML class diagram and UML swim lane activity diagram to Armani
  - UML class diagram is used to describe the static structure
    - One UML class is translated into one Armani component, may be extended with resource utilization aspect
  - UML swim lane activity diagram organizes actions and activities for classes
    - UML action states and transitions are not translated
    - UML crossing swim lane transitions are translated into Armani architectural connections, may be extended with resource utilization aspect for connectors
  - Resource utilization aspect is translated into performance properties used in Armani
Mapping algorithm for translating extended UML class diagram into Armani

**Algorithm Am.1:**

**Input:** A UML class diagram \( CD \), where \( CD.\text{Classes} = \{C_1, C_2, C_3 \ldots C_m \mid C_i \text{ is a UML class, } 1 \leq i \leq m \} \) (\(|CD.\text{Classes}| = m\));

**Output:** \( S \), which is an Armani architecture specification, where \( S.\text{Style.\text{ComponentTypes}} = \{SCT_1, SCT_2, SCT_3 \ldots SCT_n \mid SCT_i \text{ is an Armani architecture component type, } 1 \leq i \leq n\} \), \( S.\text{System.Components} = \{SC_1, SC_2, SC_3 \ldots SC_l \mid SC_i \text{ is an Armani system component, } 1 \leq i \leq l\} \), and satisfies \( m = n = l \).

**Translate** \((CD): S\)

\[
\begin{align*}
S & \leftarrow \emptyset; \\
S.\text{Style.\text{Name}} & \leftarrow \text{a randomly generated string}; \\
S.\text{System.\text{Name}} & \leftarrow \text{a randomly generated string}; \\
S.\text{System.\text{SystemResourceUtilizationAspect}} & \leftarrow CD.\text{SystemResourceUtilizationAspect}; \\
\text{while } CD.\text{Classes} \neq \emptyset & \quad \\
\text{newAmComponentType : amComponentType; } & \\
\text{newAmComponent : amComponent; } & \\
uC \in CD.\text{Classes; } & \\
\text{newAmComponentType.\text{Name} } & \leftarrow uC.\text{Name}; \\
\text{while } uC.\text{Operations} \neq \emptyset & \quad \\
\text{newAmPort : amPort; } & \\
uOp \in uC.\text{Operations; } & \\
amPort.\text{Name} & \leftarrow uOp.\text{Name}; \\
\text{newAmComponentType.\text{Ports} } & \leftarrow \text{newAmComponentType.\text{Ports} } \cup \{\text{newAmPort}\}; \\
uC.\text{Operations} & \leftarrow uC.\text{Operations } \setminus \{\text{uOp}\} \\
\text{newAmComponent.\text{Name} } & \leftarrow uC.\text{Name } + \text{“Instance”}; \\
\text{newAmComponent.\text{ComponentType} } & \leftarrow \text{newAmComponentType}; \\
\text{newAmComponent.\text{ComponentResourceUtilizationAspect} } & \leftarrow uC.\text{ComponentResourceUtilizationAspect}; \\
S.\text{Style.\text{ComponentTypes} } & \leftarrow S.\text{Style.\text{ComponentTypes} } \cup \{\text{newAmComponentType}\}; \\
S.\text{System.Components} & \leftarrow S.\text{System.Components } \cup \{\text{newAmComponent }\}; \\
CD.\text{Classes} & \leftarrow CD.\text{Classes } \setminus \{uC\}; \\
S.\text{System.\text{Style} } & \leftarrow S.\text{Style};
\end{align*}
\]

**Time complexity:** \( O(N \cdot M) \), \( N \) (number of classes), \( M \) (maximum number of operations a class has)

Algorithm for translating UML swim lane activity diagram is presented in Section 4.2.2.2 of Chapter 4 in the dissertation.
Partial Algorithm Proof:

/* Pre: uC is a UML class ∧ |uC.Operations| = X ∧ newAmComponentType is an Armani architecture component type ∧ |NewAmComponentType.Ports| = Y */

while uC.Operations ≠ ∅
    newAmPort : amPort;
    uOp ∈ uC.Operations;
    amPort.Name ← uOp.Name;
    newAmComponentType.Ports ← newAmComponentType.Ports ∪ {newAmPort};
    uC.Operations ← uC.Operations - {uOp};
/*Post: X - |uC.Operations| = |NewAmComponentType.Ports| - Y */

Proof:
1. On the initial entry to the loop:
   |uC.Operations| = X ∧ newAmComponentType is an Armani architecture component type ∧ |NewAmComponentType.Ports| = Y (from Pre), and X - |uC.Operations| = X - X = 0,
2. Suppose that P is true before an arbitrary loop iteration. Assume |uC.Operations| = A,
   |NewAmComponentType.Ports| = B, and X - A = B - Y.
   Then after the iteration:
   From NewAmComponentType.Ports ← NewAmComponentType.Ports ∪ {newAmPort};
   |NewAmComponentType.Ports| + 1 = B + 1
   From uC.Operations ← uC.Operations - {uOp}; |uC.Operations| - 1 = A - 1
   Thus, X - (A - 1) = B + 1 - Y, After the iteration, Post is true.
3. On exit from the loop:
   → X - |uC.Operations| = |NewAmComponentType.Ports| - Y = post
4. So long as the loop has not terminated,
   → |uC.Operations| > 0 → t > 0 ( t is an integer-valued function)
5. After each iteration, one element in uC.Operations is deleted from the set. This implies that |uC.Operations| (i.e., t)
   is decreased by a positive integer amount. Thus the loop will terminate, and the part of algorithm is correct.

Complete proof for the algorithm is presented in Section 4.2.2.2 of Chapter 4 in the dissertation.
Resource Utilization
FDAF Step Three: Resource utilization aspect analysis results in Armani
FDAF Approach Illustrations: Data Origin Authentication Security Aspect

- Substantive aspect
- Aspect definition
  - Adapted from data origin authentication security pattern
  - Security service that verifies an identity claimed by or for an entity
  - Static view

```java
Data Origin Authentication
priKey: String;
pubKey: String;
Algorithm: enum {RSA, DSA};
SIG: String;
GenerateKey(random-num : int, alg : Algorithm): Key
DistributeKey(receiver : String, pubKey : Key):void
EncryptData(priKey : Key, alg : Algorithm, dt : String):void
DecryptData(pubKey : Key, alg : Algorithm, dt : String):void
RetrieveSIG(data : String): SIG
```
Dynamic view

Sender: ConcreteSender

- [Initialize]: GenerateKey()
- DistributeKey()
- EncryptData()
- Request Data
- [Found Data]: RetrieveSIG()
- Send Data

Receiver: ConcreteReceiver

- DecryptData()
FDAF Step One: Extend DNS architecture design with data origin authentication aspect

DNSClient
+ SendQuestion() : void
+ AnswerArrived() : void
- ProcessAnswer() : void

Messenger
+ ClientQuestionArrived(.) : void
+ ServerAnswerArrived() : void
+ SendClientAnswer() : void
+ SendServerQuestion() : void
+ ForwardQuestion() : void
+ ForwardAnswer() : void
+ ClientQuestionAnswered() : void
+ ServerQuestionGenerated() : void

DataRefresher
+ RefreshAnswerArrived() : void
- RefreshData() : void

QueryProcessor
+ QuestionPending() : void
- DecodeMessage() : void
- EncodeMessage() : void
+ PassAnswer() : void

Database
- Initialize() : void

RequestGenerator
+ RefreshMonitorOff() : void
+ GenerateRequest() : void

RefreshMonitor
- Ready() : void
- TimerOff() : void

Client
1. QueryProcessor
2. RequestGenerator
3. Messenger
4. DataRefresher

Server
1. QueryProcessor
2. RequestGenerator
3. Messenger
4. DataRefresher

DOA Aspect
1. DecryptData()
2. GenerateKey()
3. DistributeKey()
4. EncryptData()
5. RetrieveSIG()
FDAF Step One: Extend DNS architecture design with data origin authentication aspect (continued)

Dynamic Model

**Database**
- Initialize
- GenerateKey
- DistributeKey
- EncryptData

**DataRefresher**
- RefreshAnswerArrived
- RefreshData

**RefreshMonitor**
- Ready
- TimerOff

**RequestGenerator**
- RefreshMonitorOff
- GenerateRequest

**QueryProcessor**
- QuestionPending
- DecodeMessage
- SearchAnswer
- RetrieveSIG
- Question
- Answer
- EncodeMessage
- PassAnswer
- DecodeMessage

**Messenger**
- ClientQuestionAnswered
- ServerQuestionGenerated
- ServerAnswerArrived
- ForwardAnswer
- ClientQuestionArrived
- ForwardQuestion
- Question
- Answer

**DNSClient**
- SendQuestion
- AnswerArrived
- ProcessAnswer
- DecryptData
FDAQ Step Two: Automatically generate Promela formal specification

Why formal analysis?

- Prevent inconsistency of an aspect-oriented design
  - Only “EncryptData()” is added, without “DecryptData()”, or the other way around

How?

- OCL constraints defined for the DOA security aspect by the FDAQ
- Create Promela formal specification for the design
- Use model checker Spin to discover design errors

---

Context Data Origin Authentication

|priKey: String; | pubKey: String; |
| Algorithm: enum {RSA, DSA}; | SIG: String; |
| Encryption: Boolean = false; | Decryption: Boolean = false; |
| GenerateKey(random-num : int, alg : algorithm): Key | |
| DistributeKey(receiver : String, pubKey : Key):void | |
| EncryptData(priKey : Key, alg : Algorithm, dt : String):void | |
| DecryptData(pubKey : Key, alg : Algorithm, dt : String):void | |
| RetrieveSIG(data : String): SIG | |

OCL Constraints

Context Data Origin Authentication

Invariant:

Encryption = Decryption

Context EncryptData(priKey : Key, alg : Algorithm, dt : String):void

Post:

Encryption = true

Context DecryptData(pubKey : Key, alg : Algorithm, dt : String):void

Post:

Decryption = true
FDAF Step Two: Automatically generate Promela formal specification (continued)

- **Promela**
  - Based on Hoare's communication sequential processes (CSP) language
  - Input language for model checker Spin
  - Spin: simulator, exhaustive verifier

- **Mapping form extended UML class diagram and UML swim lane activity diagram into Promela**
  - UML class diagram is used to describe the static structure
    - One UML class is translated into one Promela process type
    - OCL constraints are translated into Promela macros and then inserted as Promela assertions
  - UML swim lane activity diagram organizes actions and activities for classes
    - UML states and transitions are translated into Promela process type block statements
    - UML crossing swim lane transitions are translated into message channels between process types
Mapping algorithm for translating extended UML swim lane activity diagram into Promela (partial)

**Algorithm P.2:**
**Input:** A UML swim lane activity diagram SAD, where SAD = \{U_1, U_2, U_3 \ldots U_n | U_i is a uOrganizationUnit, 1 \leq i \leq n\}, and data origin authentication aspect DOA. The SAD has been extended with the DOA aspect;
**Output:** S, which is a PSpecification, where S.Proctypes = \{P_1, P_2, P_3 \ldots P_y | P_i is a PProctype, 1 \leq i \leq y\}, S.Channels = \{Ch_1, Ch_2, Ch_3 \ldots Ch_x | Ch_i is a pMChannels, 1 \leq i \leq x\}, and satisfies y = n;

**Translate** (SAD = \{U_1, U_2, U_3 \ldots U_n\}) : S
S : generated by Algorithm P.1;
processProtypes : integer;
processProtypes ← 0;

while SAD ≠ ∅
  u ∈ SAD;
  uName : String;
  uName ← u.Name;
  curProctype : pProctype;
  size : integer;
  size ← |S.Proctypes|;
  
  for int x ← 1 to size /*find the corresponding Pormela proctype */
    if (S.Proctypes.elementAt[x].Name == uName)
      curProctype = S.Proctypes.elementAt[x];
      processProtypes++;
      break ;
    
    if could not find such a proctype
      Input Error;

  while u.Transitions ≠ ∅
    uTran : uTransition;
    pBlock : PTBlock;
    ……

Time complexity: O(N*M), N (number of lanes), M (maximum number of transitions a lane has)
Complete algorithm is presented in Section 3.3.2.2 of Chapter 5 in the dissertation
Partial Algorithm Proof:

/* Pre: SAD is a UML swim lane activity diagram ∧ |SAD| = X ∧ processProtypes = 0 */

S : generated by Algorithm P.1;
processProtypes : integer;
processProtypes ← 0;
while SAD ≠ ∅

......

/*Post: X - |SAD| = processProtypes */

Proof:
1. On the initial entry to the loop:
   |SAD| = X ∧ processProtypes = 0 (from Pre), and X - |SAD| = X - X = 0, and processProtypes = 0 thus,
   P : X - |SAD| = processProtypes is true.
2. Suppose that P is true before an arbitrary loop iteration. Assume |SAD| = A, processProtypes = B, and X - A = B.
   Then after the iteration, two cases:
   Case (i); if (S.Proctype.elementAt[x].Name == uName):
      From processProtypes++; processProtypes = processProtypes + 1 = B + 1
      From SAD ← SAD - {u};  |SAD| = |SAD| - 1 = A - 1
      Thus, X - (A - 1) = B + 1
   Case (ii); if could not find such a proctype:
      From Input Error; The algorithm terminates;
      Thus, X - A = B
   After the iteration, Post is true.
3. On exit from the loop:
P ∧ ¬(SAD ≠ ∅) = SAD= ∅ ∧ P → X = processProtypes → X - |SAD | = processProtypes = post
4. So long as the loop has not terminated,
P ∧ (SAD≠ ∅) = X - |SAD | = processProtypes ∧ (SAD ≠ ∅) → |SAD| > 0 → t > 0 (t is an integer-valued function)
5. After each iteration, one element in SAD is deleted from the set. This implies that |SAD| (i.e., t) is decreased by a positive integer amount.
   Thus the loop will terminate, and the algorithm is correct.

Complete proof for the algorithm is presented in Section 3.3.2.2 of Chapter 5 in the dissertation
Promela Specification

#define p (Encryption == Decryption)
bool Encryption = false, Decryption = false;

int mtype = { Question, Answer }
chan DNSClientToMessenger = [0] of { mtype }
chan MessengerToQueryProcessor = [0] of { mtype }
chan MessengerToDataRefresh = [0] of { mtype }
chan RefreshMonitorToRequestGenerator = [0] of { mtype }
chan RequestGeneratorToMessenger = [0] of { mtype }

proctype DNSClient ()
{
    goto SendQuestion;
    SendQuestion: DNSClientToMessenger!Question;
    goto AnswerArrived;
    AnswerArrived: DNSClientToMessenger?Answer
    goto ProcessAnswer;
    ProcessAnswer: goto End;
    End: skip;
}

proctype Messenger ()
{
    goto ClientQuestionArrived;
    ClientQuestionArrived: DNSClientToMessenger?Question;
    goto ServerAnswerArrived;
    ServerAnswerArrived: MessengerToQueryProcessor!Answer;
    goto ProcessAnswer;
    ProcessAnswer: goto End;
    End: skip;
}
FDAF Step Three: Data origin authentication aspect analysis results in Promela

Indicates a defect in the design, a mismatch in the Encryption/Decryption mechanism. Architects need to go back to revise their design.
Summary

FDAF Aspect

Property Aspect: Performance
- Response Time
- Resource Utilization
- Throughput

Substantive Aspect: Security
- Data Origin Authentication
- Role-Based Access Control
- Log for Audit

Definition

Modeling In UML
- Swim lane activity
- Class, Swim Lane activity
- Swim lane activity
- Class
- Class
- Class

Algorithms
- Rapide
- Armani
- Æmilia
- Promela
- Alloy
- Promela

Algorithm Proofs

Tool Support

Example System
Contributions to Knowledge

- Represents non-functional properties performance and security into aspects that can be understood, modeled, and analyzed at the software architecture design level, enabling the realization of non-functional requirements for software architectures.

- Defines a UML extension, with syntax and semantics, for modeling crosscutting non-functional properties.

- Formalizes (part of) UML into a set of existing formal languages. With formal analysis tools, the automated analysis of a UML based architecture design is achieved.

- Applies the aspect concept to support the maintainability and evolvability of an architecture design with non-functional properties.
Conclusions and Future Work

- **Formal Design Analysis Framework**
  - Extensible, flexible aspect-oriented architectural framework supports the design and analysis of non-functional properties for software architectures

- **Future work**
  - Investigate interaction analysis among security aspects
  - Conform the FDAF with UML 2.0
    - Use composite structure diagram to describe software architecture
  - Extend the framework
    - Additional non-functional properties (e.g., reliability)
    - The design and analysis of product line architectures
  - Support the identifying, analysis, and negotiation among multiple, (possibly) conflicting non-functional properties
    - The Non-Functional Requirement Framework [Chung]
    - Architecture Tradeoff Analysis Method (ATAM) [Kazman]
    - Partial goal satisfaction [Letier]
Thank You!
FDAF Approach Illustrations: Response Time Performance Aspect

- **Property aspect**
- **Aspect definition**
  - Stereotype <<PAstep>> and tag PAdemand, defined in RTUML
- **DNS illustration**
  1. Extend DNS architecture design with response time aspect

<table>
<thead>
<tr>
<th>DNSClient</th>
<th>Messenger</th>
<th>QueryProcessor</th>
<th>RequestGenerator</th>
<th>RefreshMonitor</th>
<th>RefreshMonitor</th>
<th>Database</th>
</tr>
</thead>
</table>

- Stereotype <<PAstep>> and tag PAdemand, defined in RTUML

UMLState <<PAstep>>
{PAdemand = ('req', 'mean', (2,'ms'))}
Publications

- **Journal Papers**

- **Conference/Workshop Papers**
Papers in preparation

- “Aspect-Oriented Modeling and Analysis of Performance Properties in Software Architecture”, Journal on System Man Cybernet
2. Automatically generate Rapide formal specification

- **Rapide**
  - Architecture description language
  - Luckham, Stanford University
  - Evolved from VHDL, ML, and TSL
  - Provides simulation of distributed systems by partial orderings of events
  - Timing model allows designers to describe and analyze time sensitive prototypes

- **Mapping from UML class diagram and extended UML swim lane activity diagram to Rapide**
  - UML class diagram is used to describe the static structure
    - One UML class is translated into one Rapide type
  - UML swim lane activity diagram organizes actions and activities for classes
    - UML action states and the transitions between these action states are translated into Rapide type behaviors
    - UML crossing swimlane transitions are translated into Rapide architectural connections
    - UML action states are extended with response time aspect. Response time aspect is translated to Rapide’s timing model
Type DNSClient is interface
action out SendRequest();
action in AnswerArrived();
end DNSClient;

module NewDNSClient() return DNSClient is
action ProcessAnswer();
parallel
  when Start
do
    SendRequest() pause 205;
  end when;
end NewDNSClient;

Type Messenger is interface
action out ForwardAnswer(), ForwardQuestion(), SendClientAnswer(), SendServerQuestion();
action in ClientQuestionArrived(), ServerAnswerArrived(), ClientQuestionAnswered(), ServerQuestionGenerated;
end Messenger;

module NewMessenger() return Messenger is
parallel
  ClientQuestionArrived
end NewMessenger;

Response Time
Response time aspect analysis results in Rapide

Average response time of DNS server is 50 milliseconds
FDAF Approach Illustration: Throughput Performance Aspect

- Property aspect
- Aspect definition
  - Throughput for active actions
    - Actions cause the change of the system’s state
    - <<PAction>> and
    - Tag PAActionRate, rate of action
    - Tag PAActionPriority, action’s priority
    - Tag PAActionWeight, action’s execution weight
  - Throughput for passive actions
    - Actions triggered by active actions
    - <<PAction>> and
    - Tag PAreactivePriority, action’s priority
    - Tag PAreactiveWeight, action’s execution weight
DNS Illustration

1. Extend DNS architecture design with throughput aspect
2. Automatically generate \( \Phi \)milia formal specification

- \( \Phi \)milia
  - Architecture description language
  - Balsamo S.
  - Process Algebra Architecture Description Language (PADL)
  - underlying performance analysis theory: Markov Chain

- Mapping from UML class diagram and UML swim lane activity diagram to \( \Phi \)milia
  - UML class diagram is used to describe the static structure
    - One UML class is translated into one \( \Phi \)milia architectural element type (AET)
  - UML swim lane activity diagram organizes actions and activities for classes
    - UML action states and transitions are translated into \( \Phi \)milia AET behaviors
    - UML action states may be extended with throughput aspect, translated into \( \Phi \)milia’s throughput information
    - UML crossing swimlane transitions are translated into \( \Phi \)milia architectural attachments among AETs
Throughput
3. Throughput aspect analysis results in Æmilia

<table>
<thead>
<tr>
<th># of Clients</th>
<th>States</th>
<th>Transitions</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
<td>3.92283</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td>13</td>
<td>2.86861</td>
</tr>
<tr>
<td>3</td>
<td>19</td>
<td>24</td>
<td>1.59453</td>
</tr>
<tr>
<td>4</td>
<td>29</td>
<td>38</td>
<td>1.15995</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>36</td>
<td>1405</td>
<td>2070</td>
<td>0.113107</td>
</tr>
<tr>
<td>37</td>
<td>1481</td>
<td>2183</td>
<td>0.109997</td>
</tr>
<tr>
<td>38</td>
<td>1559</td>
<td>2299</td>
<td>0.107054</td>
</tr>
<tr>
<td>39</td>
<td>1639</td>
<td>2418</td>
<td>0.104264</td>
</tr>
<tr>
<td>40</td>
<td>1721</td>
<td>2540</td>
<td>0.101617</td>
</tr>
<tr>
<td>41</td>
<td>1805</td>
<td>2665</td>
<td>0.0991002</td>
</tr>
</tbody>
</table>
**FDAB Approach Illustrations: Role-Based Access Control Security Aspect**

- **Substantive aspect**
- **Aspect definition**
  - Adapted from role-based access control security pattern
  - Advanced access control because it reduces the complexity and cost of security administration
  - Concepts: user, role, right, object
  - Roles assigned to users and rights assigned to roles
  - Static view

```java
// RBACManager
RolePolicy: File
RightPolicy: File
DistributeRole(U : User) : Role
UpdateRole(U : User) : Role
RevokeRole(U : User) : boolean
CheckRole(U : User) : Role
CreateRole(F : File) : Role
DeleteRole(R : Role) : Void
AssignRight(R : Role) : Right[]
UpdateRight(R : Role) : Right[]
RevokeRight(R : Role) : Boolean
CheckRight(R : Role) : Right[]
CreateRight(F : File) : Right
DeleteRight(R : Right) : void

// User
UserId : String
UserName : String
GetUserID() : String
GetUserName() : String
GetRole() : Role[]

// Role
RoleID : String
RoleName : String
GetRoleID() : String
GetRoleName() : String

// Right
Name : String
Type : String
GetName() : String
GetType() : String

// Object
ObjectId : String
ObjectName : String
GetObjectId() : String
GetObjectName() : String
```
Dynamic view

- : ConcreteUser
- : Interface
- : AccessControl
- : ConcreteObject

- Login()
- AccessRequest()
- Forward()
- CheckRole(u)
- [Role not assigned]
- DistributeRole(u)
- [Role assigned]
- CheckRight(r)
- [Permitted]
- Access()
DNS Illustration

1. Extend DNS architecture design with role-based access control aspect

```
CreateRole()
DistributeRole()
CreateRight()
AssignRight()
```

---

```
DNSClient
+ SendQuestion() : void
+ AnswerArrived() : void
- ProcessAnswer() : void

Messenger
+ ClientQuestionArrived() : void
+ ServerAnswerArrived() : void
+ SendClientAnswer() : void
+ SendServerQuestion() : void
+ ForwardQuestion() : void
+ ForwardAnswer() : void
+ ClientQuestionAnswered() : void
+ ServerQuestionGenerated() : void

QueryProcessor
+ QuestionPending() : void
- DecodeMessage() : void
- SearchAnswer() : void
- EncodeMessage() : void
+ PassAnswer() : void

RequestGenerator
+ RefreshMonitorOff() : void
+ GenerateRequest() : void

DataRefresher
+ RefreshAnswerArrived() : void
- RefreshData() : void

Database
- Initialize() : void

Server

Client

query
searches
refresh message
execution
<<after>>

RBAC Aspect
CreateRole()
DistributeRole()
CreateRight()
AssignRight()

RefreshMonitor
- Ready() : void
+ TimerOff() : void

refresh message
refreshes
monitors
timer off
2. Automatically generate Alloy formal specification

- **Why formal analysis?**
  - Prevent inconsistency of role-based access control policies

- **How?**
  - Access control policies defined in OCL for the DNS role-based access control model
  - Create Alloy formal specification for the design
  - Use Alloy’s analysis tool to discover design errors

```plaintext
Context RBAC inv
  self.User -> size() = 7;
  self.Role -> size() = 4;
  self.Right -> size() = 6;
  self.User -> forall(u | u.Role -> size() > 0);
  self.User -> select(u | u.Role -> size() = 1) -> size() = 2;
  self.User -> forall(u | u.Role -> size() < 3);
  self.Role -> forall(r | r.Role -> size() > 0);
  self.Role -> forall(r | r.Role -> size() < 4);
  self.Role -> one(r | r.Role -> size() = 1);
  self.Role -> forall(r | r.Right -> size() = 1);
  self.Right -> forall(r | r.Role -> size() > 0);
  self.Right -> forall(r | r.Role -> size() > 1 and r.Role -> size() < 4);
  self.Role -> forall(r | r.Right -> size() < 4);
  self.Role -> forall(r1, r2 | r1 <> r2 implies r1.Right - r2.Right -> NotEmpty());
  self.User -> forall(u | u.Role.Right -> size() < 3);
```
Alloy
- Software Design Group
- Based on set theory and first order logic
- Analysis tool is a constraint solver that provides automatic simulation and checking
- Model finder

Mapping form extended UML class diagram with OCL into Alloy
- UML class diagram is used to describe the static structure
  - UML classes and interfaces are translated to Alloy signatures
  - A UML association is translated into a relationship (or a filed) for the Alloy signature
- OCL invariants are translated to equivalent Alloy formulae UML
<table>
<thead>
<tr>
<th>OCL</th>
<th>Equivalent Alloy Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-&gt; size() Op y</td>
<td>#X Op y or #(X) Op y</td>
</tr>
<tr>
<td>X-&gt; forall(x</td>
<td>P(x))</td>
</tr>
<tr>
<td>X-&gt; exists(x</td>
<td>P(x))</td>
</tr>
<tr>
<td>X-&gt; one(x</td>
<td>P(x))</td>
</tr>
<tr>
<td>X-&gt; notEmpty()</td>
<td>some X</td>
</tr>
<tr>
<td>X-&gt; isEmpty()</td>
<td>no X</td>
</tr>
<tr>
<td>X-&gt; select(x</td>
<td>P(x))-&gt; size() = 1</td>
</tr>
<tr>
<td>X-&gt; select(x</td>
<td>P(x))-&gt; size() = y (y&gt;1)</td>
</tr>
<tr>
<td>X-&gt; select(x</td>
<td>P(x))-&gt; size() &gt; y (y&gt;0)</td>
</tr>
<tr>
<td>X-&gt; select(x</td>
<td>P(x)) -&gt; notEmpty()</td>
</tr>
<tr>
<td>X-&gt; select(x</td>
<td>P(x)) -&gt; isEmpty()</td>
</tr>
<tr>
<td>X-&gt; union(Y)</td>
<td>X + Y</td>
</tr>
<tr>
<td>X-&gt; intersection(Y)</td>
<td>X &amp; Y</td>
</tr>
<tr>
<td>X-Y</td>
<td>X-Y</td>
</tr>
<tr>
<td>P and Q</td>
<td>P &amp;&amp; Q</td>
</tr>
<tr>
<td>P or Q</td>
<td>P</td>
</tr>
<tr>
<td>P implies Q</td>
<td>P =&gt; Q</td>
</tr>
<tr>
<td>a &lt;&gt; b</td>
<td>a != b</td>
</tr>
<tr>
<td>a = b</td>
<td>a = b</td>
</tr>
</tbody>
</table>
Fact Definitions

```
AssignedURoles = "AssignedUsers
AssignedRights = "AssignedRoles
```

Fact Policy

```
//Policy 1: There are seven users in the RBAC model.
\# (User) = 7

//Policy 2: There are four roles in the RBAC model.
\# (Role) = 4

//Policy 3: There are six rights in the RBAC model.
\# (Right) = 6

//Policy 4: Each user should have at least one role.
all u : User | \#u,AssignedURoles > 0

//Policy 5: Exactly two users have only one role.
some u1 : User | \#u1,AssignedURoles = 1 &
one u2 : User - u1 | \#u2,AssignedURoles = 1
```
3. Role-based access control aspect analysis results in Alloy

```plaintext
fact Policy {
  // There are 7 users
  # (User) = 7

  // There are 4 roles
  # (Role) = 4

  // There are 6 rights
  # (Right) = 6
```

Access Policies

#1: 7 users
#2: 4 roles
#3: 6 rights
#4: Each user should have at least one role

#5: Exactly two users have only one role

#6: No user has more than two roles
#7: All roles have been assigned
#8: No role can be assigned to more than half the users

Solutions

AssignedURoles : some modes/examples/tutorials/RBAC/Role =
{User_0 -> Role_0, User_1 -> Role_1,
User_2 -> {Role_2, Role_3}, User_3 -> {Role_2, Role_3},
User_4 -> {Role_2, Role_3}, User_5 -> {Role_2, Role_3},
User_6 -> {Role_2, Role_3}}

AssignedUsers : some modes/examples/tutorials/RBAC/User =
{Role_0 -> {User_0, User_1, User_2},
Role_1 -> {User_1, User_3, User_4},
Role_2 -> {User_2, User_5, User_6},
Role_3 -> {User_3, User_4, User_5}}
Access Policies

#9: There is one role that can only be assigned to one user

Inconsistency: Policy #5 and Policy #9 together over-constraint the model

#5 Updated from “Exactly two users have only one role” to “At least two users only has one role”

#10: All roles should have at least one right
#11: All rights have been assigned
#12: One right should be assigned to at least two roles, but at most three roles
#13: No role should have more than half the rights
#14: Different roles should not have the same set of rights
#15: No user can have more than half the rights

Solutions

Compiling RBAC.als ...
Compilation successful.
Executing command run Realism for 2 but 17 modes/examples/tutorials/RBAC/Object...
Instance found: Realism is consistent. (00:50)
Compiling RBAC.als ...
Compilation successful.
Executing command run Realism for 2 but 17 modes/examples/tutorials/RBAC/Object...
No instance found: Realism may be inconsistent. (00:49)

......
\{User_0 \rightarrow Role_0, User_1 \rightarrow Role_0, User_2 \rightarrow Role_1, 
User_3 \rightarrow Role_2, User_4 \rightarrow Role_2, User_5 \rightarrow Role_3, 
User_6 \rightarrow Role_3\}

......
\{Role_0 \rightarrow \{Right_0, Right_1, Right_5\}, 
Role_1 \rightarrow \{Right_2, Right_3, Right_4\}, 
Role_2 \rightarrow \{Right_2, Right_3, Right_5\}, 
Role_3 \rightarrow \{Right_0, Right_1, Right_4\}\}

......
\{Role_0 \rightarrow \{User_0, User_1\}, Role_1 \rightarrow User_2, 
Role_2 \rightarrow \{User_3, User_4\}, Role_3 \rightarrow \{User_5, User_6\}\}

......
\{Right_0 \rightarrow \{Role_0, Role_3\}, Right_1 \rightarrow \{Role_0, Role_3\}, 
Right_2 \rightarrow \{Role_1, Role_2\}, Right_3 \rightarrow \{Role_1, Role_2\}, 
Right_4 \rightarrow \{Role_1, Role_3\}, Right_5 \rightarrow \{Role_0, Role_2\}\}
Role assignment and right assignment for DNS RBAC Model

- Users: Ann (User_0), Cathy (User_1), John (User_2), Mark (User_3), Mary (User_4), Paul (User_5), Seth (User_6)
- Roles: Administrator (Role_1), Developer (Role_2), Tester (Role_3), User (Role_0)
- Rights: View (Right_0), Report (Right_1), Debug (Right_2), Delete (Right_3), Shutdown (Right_4), Modify (Right_6)
Substantive aspect

Aspect definition

- Adapted from log for audit security pattern
- Systematic evaluation of company information security measuring to established criteria
- Ties logging to auditing, to ensure that logging is configured with audit in mind and auditing is understood to be integral to effective logging
- Static view

Log for Audit Security Aspect

- InitializeLoggingAPI(api : LoggingAPI):void
- MethodCallTracing(msg : LogMsg, file : LogFile):void
- ExceptionLogging(msg : LogMsg, file : LogFile):void
- TimeProfiling(msg : LogMsg, file : LogFile):void
- MemoryProfiling(msg : LogMsg, file : LogFile):void
- UserActionLogging(msg : LogMsg, file : LogFile):void
- RecoveringLostData(file : LogFile) : void
- FormatLogStatements(file : LogFile):void
- LogAnalyzer(file : LogFile):void
- LogFileSizeMonitor():void
- LogLogic():void
DNS Illustration

1. Extend DNS architecture design with log for audit aspect

```
DNSClient
+ SendQuestion() : void
+ AnswerArrived() : void
- ProcessAnswer() : void

Messenger
+ ClientQuestionArrived() : void
+ ServerAnswerArrived() : void
+ SendClientAnswer() : void
+ ForwardQuestion() : void
+ ForwardAnswer() : void
+ ClientQuestionAnswered() : void
+ ServerQuestionGenerated() : void

QueryProcessor
+ QuestionPending() : void
+ PassAnswer() : void

RequestGenerator
+ RefreshMonitorOff() : void
+ GenerateRequest() : void

DataRefresher
+ RefreshAnswerArrived() : void
- RefreshData() : void

Database
- Initialize() : void

RefreshMonitor
- Ready() : void
+ TimerOff() : void

Client

Server

refresh message

Monitors

Log for Audit Aspect

TimeProfiling()

execution

<before> & <after>

Log for Audit Aspect

TimeProfiling()

Execution

<before> & <after>

Log for Audit Aspect

TimeProfiling()

Timer Off
```
Why formal analysis?
- Prevent inconsistency of an aspect-oriented design
  - “TimeProfiling” and “MemoryProfiling” should crosscut a UML operation twice (before and after)

How?
- OCL constraints defined for the log for audit aspect
- Create Promela formal specification for the design
- Use model checker Spin to discover design errors

---

Log for Audit
- Initialization: Boolean = false
- LoggingAPI: Logging API
- LogMsg: String
- LogFile: File[]
- InitializeLoggingAPI(api : LoggingAPI):void
- MethodCallTracing(msg : LogMsg, file : LogFile):void
- ExceptionLogging(msg : LogMsg, file : LogFile):void
- TimeProfiling(msg : LogMsg, file : LogFile, time : String):void
- MemoryProfiling(msg : LogMsg, file : LogFile, memory : String):void
- UserActionLogging(msg : LogMsg, file : LogFile):void
- RecoveringLostData(file : LogFile):void
- FormatLogStatements(file : LogFile):void
- LogAnalyzer(file : LogFile):void
- LogFileSizeMonitor():void
- LogLogic():void

---

Context Log for Audit Aspect
Invariant: Initialization = true
Context LFA::InitializeLoggingAPI(api : LoggingAPI):void
pre: Initialization = false
post: Initialization = true
Context Package
Invariant:
let TPCrosscutting := self.allCrosscutting ->
select(cross : self.allCrosscutting |
cross.crosscuts = TimeProfiling or MemoryProfiling)
TPCrosscutting -> forall (TP : TPCrosscutting |
TimeProfiles -> includes
(symmetricCrosscutting(TP)))
Emtype = { Question, Answer }; 
chan DINSClientToMessenger = [0] of {mtype}; 
chan MessengerToQueryProcessor = [0] of {mtype}; 
chan MessengerToDataRefresher = [0] of {mtype}; 
chan RefreshMonitorToRequestGenerator = [0] of {mtype}; 
chan RequestGeneratorToMessenger = [0] of {mtype};

proctype DINSClient ()
{
    goto SendQuestion;
    SendQuestion; DINSClientToMessenger?Question;
    goto AnswerArrived;
    AnswerArrived; DINSClientToMessenger?Answer;
    goto ProcessAnswer;
    ProcessAnswer; goto End;
    End; skip;
}

proctype Messenger()
{
    goto ClientQuestionArrived;
    ClientQuestionArrived; DINSClientToMessenger?Question;
    goto ForwardQuestion;
    ForwardQuestion; MessengerToQueryProcessor?Question
}
3. Log for audit aspect analysis results in Promela

Indicates a defect in the design, a mismatch in using profiling operations. Architects need to go back to revise their design.
**Object-Oriented Design**

- Problem Statements
- Module 1
  - Security
  - Performance
  - Adaptability
- Final Design

**Problem:** lower cohesion, higher coupling, maybe significant re-work when facing changes

**Aspect-Oriented Design**

- Problem Statements
- Module 1
  - Aspects added one at a time
  - Security
  - Adaptability
- weaver
- Final Design

**Advantages:** Tangling aspects handled separately, Dramatically reduce the complexity of understanding, change, and analysis, promoting reusability
Rapide Timing Model

- A clock is a monotonically increasing counter
- The rate at which it increases is not related to any physical unit but rather is controlled by a computation
- An increase in time as a tick. Clocks tick when there are no more events to be generated at the current time
- An event may be timed with respect to one or more clocks.
- Events have start and finish times modeling event duration
Armani Performance Analysis Theory

- Queuing network theory
- Little’s law
  - $<<\text{NetworkPop>>}.\text{value} = <<\text{ArrivalRate>>}.\text{value} \times <<\text{NetworkResponse>>}.\text{value}$
  - M/M/1 queue
    - $<<\text{Utilization>>}.\text{value} = <<\text{ArrivalRate>>}.\text{value} \times <<\text{Visits>>}.\text{value} \times <<\text{ServiceTime>>}.\text{value}$
    - $<<\text{ResponseTime>>}.\text{value} = <<\text{ServiceTime>>}.\text{value} / (1 - <<\text{Utilization>>}.\text{value})$
    - $<<\text{Length>>}.\text{value} = <<\text{ArrivalRate>>}.\text{value} \times <<\text{Visits>>}.\text{value} \times <<\text{ResponseTime>>}.\text{value}$

- Adaptation at the software architecture level
  - $<<\text{NetworkRespone>>}.\text{value} = \sum_{i=1}^{n} \text{Component}_i. <<\text{ResponseTime>>}.\text{value} \times \text{Component}_i. <<\text{Visits>>}.\text{value} + \sum_{j=1}^{n} \text{Connector}_j. <<\text{Visits>>}.\text{value} \times \text{Connector}_j. <<\text{DelayTime>>}.\text{value}$
  - $<<\text{NetworkPop>>}.\text{value} = <<\text{ArrivalRate>>}.\text{value} \times <<\text{NetworkResponse>>}.\text{value}$
Markov chain

\[
P[\chi(t) = x \mid \chi(t_n) = x_n, \chi(t_{n-1}) = x_{n-1}, \ldots, \chi(t_0) = x_0] = P[\chi(t) = x \mid \chi(t_n) = x_n]
\]

Reward structures

- A yield function \( y_{i,j}(t) \) expressing the rate at which reward is accumulated at state \( i \) \( t \) time units after \( i \) was entered when the successor state is \( j \)
- A bonus function \( b_{i,j}(t) \) expressing the reward awarded upon exit from state \( i \) and subsequent entry into state \( j \) given that the holding time in state \( i \) was \( t \) time units
Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;

Entity CONTROL Is
    Port ( 
        Reset: in std_logic;
        Clk: in std_logic;
        Mode: in std_logic;

        ......);
End CONTROL;

Architecture CONTROL_A of CONTROL Is
    constant FRAMESIZE: integer := 253243;
    constant TESTADDR: integer := 253000;
    signal ENDFR: std_logic;
    signal INCAD: std_logic;
    signal VS: std_logic;

    Begin
        ADDRCTR: process(Clk)
            variable cnt: unsigned (17 downto 0);
            begin
                if rising_edge(Clk) then
                    if TestLoad = '1' then cnt := to_unsigned(TESTADDR,18); ENDFR <= '0';
                        else
                        ......
                End CONTROL_A;
Standard ML Example (Mergesort)

- fun merge([], y) = y
  = | merge(x, []) = x
  = | merge(a::x, b::y) =
  = if a < b then a::merge(x, b::y)
  = else b::merge(a::x, y);

val merge = fn : int list * int list -> int list - merge([1,4,5],[2,3,4]);
val it = [1,2,3,4,4,5] : int list
Process Algebra

\[\begin{align*}
\text{Sender} & := \mu \alpha . m!.a?.\alpha \\
\text{Receiver} & := \mu \beta . m?.a!.\beta \\
\text{System} & := (\nu m)(\nu a)(\text{Sender} \mid \text{Receiver})
\end{align*}\]
CSP Example (Consumer & Producer)

```
Buffer::
  buf (0 .. bufsize-1) : buffer-element;
  first, last       : integer;   -- queue pointers
  j, k             : integer;   -- process counters
  first := 0;
  last := 0;
  *[ (j: 1..numbprod)
      (last+1) mod bufsize ≠ first;   -- if there is room in the buffer,
      Producer(j) ? buf(last) →   -- read an element.
      last := (last + 1) mod bufsize
  ]

  (k: 1..numbcons)
  first ≠ last;   -- if there is something in the buffer
  Consumer(k) ? more() →   -- and a consumer signals a desire to consume,
  Consumer(k) ! buf(first);   -- send that consumer an element.
  first := (first + 1) mod bufsize ]

-- The buffer runs concurrently with the producers and consumers.
PRODUCER is the text of the producer processes; CONSUMER, of the consumer processes.
|| (i: 1..numbprod) PRODUCER
|| (i: 1..numbcons) CONSUMER ]
```